**Description (Short)**

To solve the minimum cost bipartite matching problem, we firstly convert the graph into a minimum cost flow problem by forming an equivalent graph, solve the problem through cycle cancellation algorithm and then extracting the edges with non-zero flow. We then declare them as minimum cost edges.

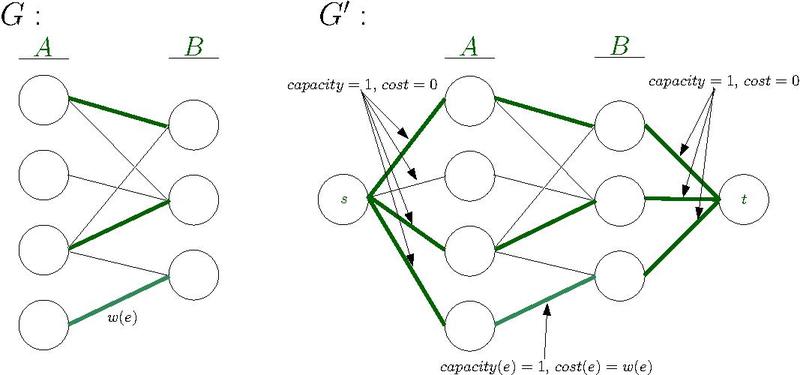
**Description (Long)**

Given a bipartite graph, our objective is to find out a minimum cost perfect matching. This can be done by solving an equivalent minimum cost flow problem and then choosing the edges with non- zero flow. The chosen edges would correspond to the minimum cost matching.

1. **Conversion to minimum cost flow problem**

Let the given bipartite graph be G= (A ∪ B, E) where A represents the set of nodes on the first connected components, B represents the set of nodes on the second connected components and E represents the set of edges. The edges in E are supposed to satisfy the condition that the outgoing node lies in A and the outgoing node lies in B.

Let the corresponding graph for minimum cost flow problem be G’ = (A ∪ B ∪ (source [s], sink [t]), E). Now each edge in the set E has a capacity 1 and cost as the same in graph G. the graphs. Now we also augment edges from source[s] to every node in A and from every node in B to sink[t]. The capacity of all these edges is 1 and the cost is zero.



1. **Cycle Cancellation**

Now that we have found the equivalent graph G’, we solve the minimum cost flow problem for it.

The objective is given the graph G’, a source node ‘s’, a sink node ‘t’ and a given amount of flow ‘f’, we need to find out the flow through each edge of a graph such that

1. minimum cost is achieved and
2. outflow of source = outflow of sink = f

The whole process can be broken down into two stages. Firstly, we initialize a flow ‘f’ in the graph using the Edmond Karps algorithm. Note that at this stage, the flow is not optimal in terms of cost.

Secondly, we generate a residual network of the graph. Then we find a negative cost cycle using Edmond Karps algorithm and then find the minimum flow which we can add to this cycle. The residual graph is updated accordingly and then the second step is repeated until there exists no negative cycle in the graph. This process ultimately gives the optimal flow in terms of cost.

The pseudocode is as follows:

Cycle Cancellation( Graph: G, source node: s, sink node: t, flow: f):

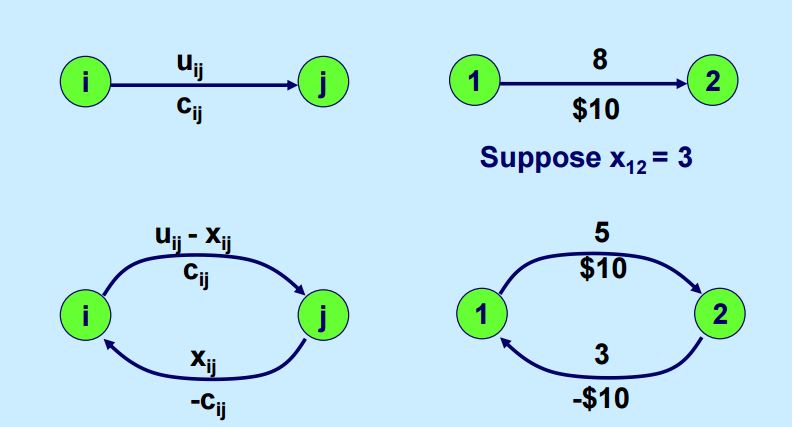
* Establish a feasible flow of ‘f’ in the network: FlowG
* Generate Residual network Gres
* while ( Gres has a negative cost cycle):
  + Compute the negative Cost cycle C
  + Find the minimum flow edge in C = ‘d’
  + For edges in G and C : add ‘d’ to the FlowG
  + For edges in Gres and C: subtract d from the flow (Update to residual network)
* Return FlowG
  1. **Flow Initialization (Edmond Karps Algorithm)**

Given f’ is the maximum flow in a graph G, the argument as G and flow f, Edmond Karps algorithm returns a feasible flow f in the graph G if f < f’ else returns the maximum possible flow f’. Initially the flow is made to be zero and then we keep finding augmenting paths in the residual graph until there are none left and then add the minimum capacity edge’s capacity in the path to all edges in the path.

* 1. **Residual Networks**

We generate the residual network by the following rule. If there is an edge from V1 to V2, with cost c12, capacity u12 and current flow x12, then the residual graph has an edge

* + 1. From V1 to V2, with capacity u12 – x12, cost c12
    2. From V2 to V1, with capacity x12 and cost –c12



Source: MIT OCW Network optimization Fall 2010

* 1. **Negative Cycles (Bellman Ford)**

To compute the negative cycles in the graph we use bellman ford algorithm which is basically used to compute minimum distance path between two vertices. It runs in O(|V||E|) time. The algorithm is described as follows:

Firstly, we obtain a list of distances and predecessors to each node, initialized to infinity and null respectively.

Then for |V| -1 iterations, we ‘relax’ edges repeatedly i.e. update the distance to a node and its predecessor if the distance to it can be shortened by taking a different edge. Then we check for negative cycles by canning all the edges and by finding a path which has length |V| which can only happen if atleast one negative cycle exists in the graph.

**Results**

1. For a working example, we took the following function:

F(a,b,c,d,e,f,z) = [{(a OR b) OR (c AND e)} XOR (d OR z)] OR f

The corresponding expression tree is as follows:

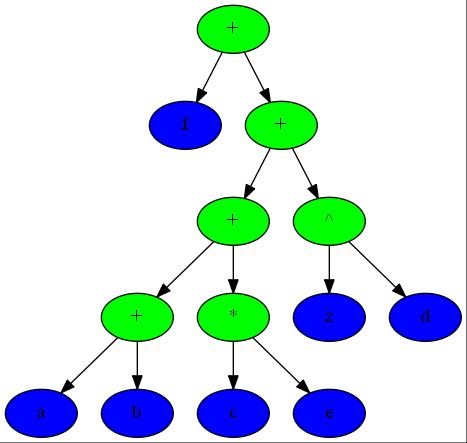


Figure: Expression Tree of the Function F(a,b,c,d,e,f,d,z). The green nodes represent an operation and the blue nodes represent a variable.

1. Next, to form the corresponding we allot one node each to a variable and an operation (OR, AND, XOR). The optimal locations (keeping the locations of ‘variable’ nodes predetermined) of the ‘operator nodes’ in terms of the length of wire are found out using the Linear Programming Formulation of the problem. The following are the parameters used:
   1. Minimum Seperation between two nodes : 0.1
   2. Cost per unit length (Alpha) = 1

Using these parameters, we obtain the initial position of nodes as follows:

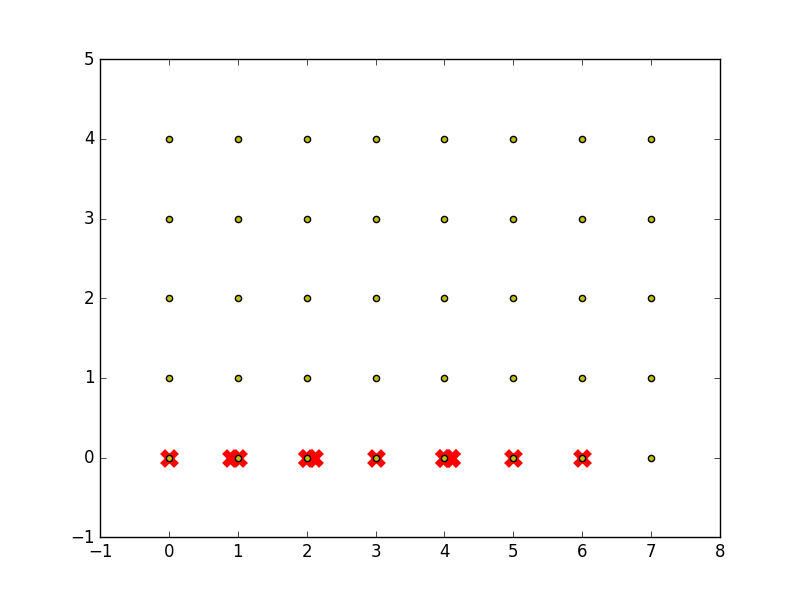
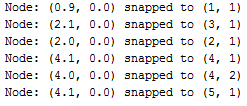


Figure: Initial Node positions are represented by a red cross. Yellow dots are the discrete positions available on the graph. Note that all these nodes don’t lie on the discrete positions yet.

1. Now we turn the ‘snapping’ of nodes problem into a bipartite weighted minimum cost matching problem. Then we solve it using Cycle Cancellation algorithm and obtain the following discretization of the position of nodes.



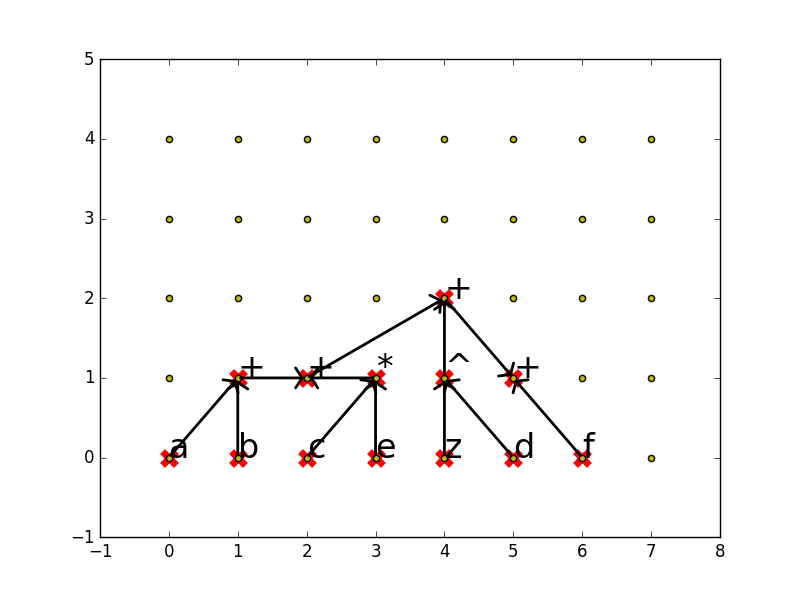


Figure: Grid showing discretized locations of each node (represented as a red cross). The arrows represent the fanin and fanout of each node.